

Assignment 2: Creating ADTs, Implementing DS and an XML Parser

Description

In this assignment, you'll work in groups to write your own ADTs for a stack and a queue, and then create working versions of an array list, a double linked list, a stack and a queue. You'll then use these utility classes to implement your XML document parser, which will read and confirm that the XML files are properly formatted.

Equipment and Materials

For this assignment, you will need:

- Eclipse, Java 8 and JUnit 4

Instructions

This assignment consists of multiple parts, to be completed **outside** of class time. See the course outline and Brightspace for due dates. Complete this assignment with your assigned group.

Part A: Create and Implement an XML Parser using your own data structures (90 marks)

1. Using the requirements and hints in the *Specifications* section below:
 - a. Create your own stack and queue interface classes.
 - b. Implement your utility classes for an array list, double linked list, a stack and a queue.
 - c. Use the utility classes to implement an XML document parser.
2. There are **two** parts with different submission folders for this assignment. The requirements for each part are outlined in the *Submission Deliverables* section of this document. Each of the two submissions has a **separate due date**. See Brightspace for exact due dates.

Part B: Complete an Evaluation as a Group (5 marks)

After completing your XML parser application, check your work against the provided marking criteria. Your instructor will refer to your group's self-evaluation when grading the assignment and will provide further feedback and grade adjustments as needed. Your instructor is responsible for awarding the group's final grade.

1. Open the Marking Criteria documents (MarkingCriteria_Assignment2.docx) and save a copy with your group's name.

2. As a group, discuss how well you met each criterion and assign yourselves a mark for each row in the table. You may include a short, point form, explanation for your mark in the Notes column.

Note: This is an opportunity for you to identify any missing pieces according to the marking criteria, make sure you do this step diligently to avoid losing any marks that is clearly stated in this document.

3. Save this file for submission to Brightspace along with your completed code.

Part C: Complete a Peer Assessment (5 marks)

Each student must also complete a peer assessment of their group members. Your instructor will provide further submission details.

Assignment Specifications

Important: Read the specifications very carefully. If you are uncertain about any of the requirements, discuss it with your instructor.

1. Write your own **StackADT.java** and **QueueADT.java** interface with all the required functionalities as method stubs with appropriate pre-conditions, post-conditions, parameters, return values and expected exceptions using proper Javadoc notations.

Note: These 2 files are the only requirements for part 1 submission.

2. Do NOT wait to start this next step!
3. Import the assignment2StartingCode project into Eclipse.
4. Write an implementation for the utility class **MyArrayList.java** using the supplied **ListADT.java** and **Iterator.java** interfaces using an array as the underlying data structure to store the elements in this list.

Note: Do NOT modify the ADTs provided in any way! Your instructor will test with the default ADT files.

5. Completely test the implementation of **MyArrayList.java** for correct functionality using the set of JUnit tests from the starting code.

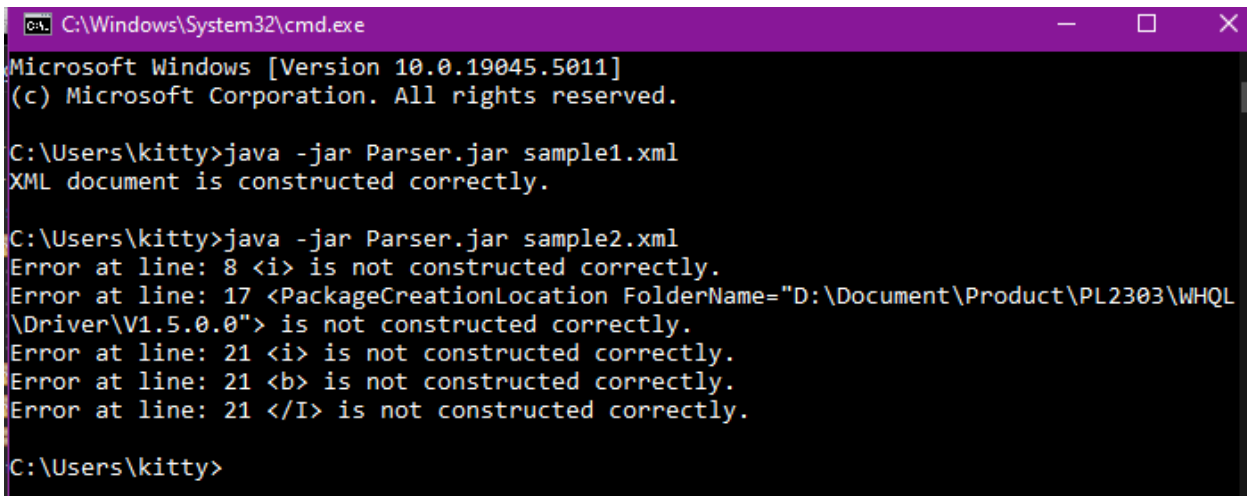
Note: Do NOT modify the tests provided in any way! Your instructor will test with the default test files.

6. Repeat steps 3–4 for the utility class **MyDLL.java** using a linked list as the underlying data structure to store each element in a node, based on the same **ListADT.java** and **Iterator.java** interfaces provided by your instructor.
 - a. Your implementation should also include a **MyDLLNode.java** class to store each individual element in the DLL.

7. After the due date of part 1, part 2 will be released with the ADTs you will need for the following steps.
8. Write an implementation for the utility class **MyStack.java** using the **StackADT.java** and **Iterator.java** interfaces provided by your instructor. Use your **MyArrayList.java** implementation as the underlying data structure to store the elements of the stack.
9. Write an implementation for the utility class **MyQueue.java** using the **QueueADT.java** and **Iterator.java** interfaces and **EmptyQueueException.java** provided by your instructor. Use your **MyDLL.java** implementation as the underlying data structure to store the elements of the queue.

Important: Move to the next step only when you are satisfied that the data structures above perform properly. Then implement your XML document parser as described below.

10. Write an XML parser that will accomplish the following:
 - a. Read supplied XML documents.
 - b. Parse for errors in the XML construction.
 - c. When parsing is complete, print all lines that are not properly constructed in the order in which the errors occur.
11. Ensure that your program can be supplied the XML document via the command line and show all results of the parsing on the console. E.g.:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kitty>java -jar Parser.jar sample1.xml
XML document is constructed correctly.

C:\Users\kitty>java -jar Parser.jar sample2.xml
Error at line: 8 <i> is not constructed correctly.
Error at line: 17 <PackageCreationLocation FolderName="D:\Document\Product\PL2303\WHQL
\Driver\V1.5.0.0"> is not constructed correctly.
Error at line: 21 <i> is not constructed correctly.
Error at line: 21 <b> is not constructed correctly.
Error at line: 21 </I> is not constructed correctly.

C:\Users\kitty>
```

Note: The above does not show the output from the test files provided, it is simply to give you an example of what the input to the application is and what the output would look like – your results will vary depending on the file being parsed. Your program output should be similar but does NOT have to match the exact formatting as long as the relevant information are shown. Your instructor will use a different test data file to evaluate your application.

Hints

An XML document is syntactically correct if it meets the following requirements:

- An opening tag has the format `<tag>` and a closing tag has the format `</tag>`.
Note: Although, XML tags may contain attributes in the format of `name="value"` pairs, **ignore** these attributes for this assignment.
- For every closing tag, there is an earlier, matching opening tag.
- An exception to the above is a self-closing tag, which has the format `<tag/>`. Self-closing tags require no closing tag.
- The sub-phrase between a pair of matching tags is well-constructed.
- All tags are case-sensitive.
- Every XML document must have one and only one root tag.
- Tags in the format: `<?xml somedata="data"?>` are processing instructions and can be **ignored** for this assignment.
- If nested, the tag pairs cannot intercross. For example, the following is not syntactically correct:
`This is to be bold and <i>italic</i>`

Warning!

You will be penalized **50% of your final mark for the assignment** if you use the java classes from the `java.util.*`, or similar packages for the implementations of `MyStack`, `MyQueue`, `MyArrayList` and `MyDLL`.

The exception is using the `Arrays.copyOf()` or `System.arraycopy()` methods to resize your array in `MyArrayList`, the `toArray` methods and the `Iterator` implementations. You are also allowed to use any of the standard exceptions in the Java library such as `NullPointerException` and `NoSuchElementException`.

This restriction only applies for the 4 data structures, you may use any Java libraries for the XML parser component.

Submission Deliverables

Note: This is a group assessment, so only one submission is required per group. If there are multiple submissions, your instructor will only evaluate the last submission. See Brightspace for the exact due date and time.

Part 1 Submission: Stack and Queue ADTs (8 marks)

Your group's submission should include a zipped folder named as the assignment number and your group name (e.g., A2P1Zelda.zip). The zipped folder should contain **ONLY** the following 2 items:

1. Your StackADT.java
2. Your QueueADT.java

Note: Although you do not need to submit it at this time, you may want to review the Marking Criteria document to make sure you have completed this submission correctly.

Part 2 Submission: Data Structures and XML Parser (87 marks)

Your group's submission should include a zipped folder named as the assignment number and your group name (e.g., A2SPZelda.zip). The zipped folder should contain the following items:

- 1) An executable Java Archive file (.jar) for your sort application called **Parser.jar**.
- 2) A **readMe.txt** file with instructions on how to install and use the XML Parser program.
- 3) The project should have completed javadoc using the "-private" option when generated, and the output placed in the **doc** directory of the project.
- 4) A folder containing the complete Eclipse project directory. Refer to the exact instructions of how to export your project from Lab 0.
- 5) The completed Marking Criteria document containing your group's evaluation of your application. All group members should be present when completing this document and it acts as your "sign-off" (approval) of the assignment submission.

No late assignments will be accepted.